ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Discrete Applied Mathematics 148 (2005) 63–87

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Optimal parallel machines scheduling with availability constraints

Anis Gharbi[a, b], Mohamed Haouari[a]

[a] *Combinatorial Optimization Research Group - ROI, Ecole Polytechnique de Tunisie, BP 743, 2078, La Marsa, Tunisia*

[b] *Department of Applied Mathematics, Institut Supérieur d'Informatique, Ariana, Tunisia*

Received 16 April 2003; received in revised form 30 October 2004; accepted 16 December 2004
Available online 20 January 2005

Abstract

We address a generalization of the classical multiprocessor scheduling problem with non simultaneous machine availability times, release dates, and delivery times. We develop new lower and upper bounds as well as a branching strategy which is based on a representation of a schedule as a permutation of jobs. We show that embedding a semi-preemptive lower bound based on max-flow computations in a branch-and-bound algorithm yields very promising performance. Computational experiments demonstrate that randomly generated instances with up to 700 jobs and 20 machines are solved within moderate CPU time. Moreover, the versatility of the proposed approach is assessed through its ability to solve large instances of two important particular cases $P, NC_{inc}||C_{\max}$ and $P|r_j, q_j|C_{\max}$.

Keywords: Parallel machines; Machine availability; Release date; Delivery time; Branch-and-bound

1. Introduction

In this paper, we consider the problem of scheduling a set J of n jobs on m identical parallel machines ($n > m \geqslant 2$) with respect to availability restrictions for both jobs and machines. Each job j ($1 \leqslant j \leqslant n$) has a processing time p_j, a release date (or head) r_j on which the job becomes available for processing, and a delivery time (or tail) q_j that must

E-mail address: mohamed.haouari@ept.rnu.tn (M. Haouari).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2004.12.003

elapse between its completion on the machine and its exit from the system. Each machine M_i $(1 \leqslant i \leqslant m)$ has an availability time a_i on which it becomes continuously ready for working. All data are assumed to be positive integers and deterministic. A *schedule* σ is defined by an assignment of the jobs to the machines and a vector $(t_1, t_2, \ldots, t_n)$, where t_j denotes the start time of job j. A schedule is said to be *feasible* if each job is processed, with no interruption, by exactly one machine and each machine processes at most one job at one time. In addition, for $1 \leqslant j \leqslant n$ and $1 \leqslant i \leqslant m$ the following property must hold:

$$\text{Job } j \text{ is assigned to machine } M_i \Longrightarrow t_j \geqslant \max(a_i, r_j).$$

Such a schedule induces a well defined makespan $C_{\max}(\sigma) = \max_{1 \leqslant j \leqslant n} (t_j + p_j + q_j)$. The problem is to find a feasible schedule of minimum makespan. In the standard three field scheduling notation of Graham et al. [8], this problem is denoted by $P, NC_{inc}|r_j, q_j|C_{\max}$, where NC_{inc} indicates that the number of available machines is nondecreasing with time [23]. Although, many of its particular cases have been extensively studied, to the best of our knowledge this problem has not previously been addressed in the scheduling literature.

It is easy to show that $P, NC_{inc}|r_j, q_j|C_{\max}$ can be restated as an equivalent $P, NC_{inc}|r_j|L_{\max}$ (i.e. minimizing the maximum lateness on identical parallel machines with non-simultaneous machine available times and release dates). This latter models the following situation. Jobs enter the system in batches on a periodic basis. Prior to processing, job j $(j = 1, \ldots, n)$ requires a setup time r_j and should be ideally completed before its due date d_j. At the start of the planning period, some machines may not be available because they are still processing the previous batch and we want to begin scheduling the new arriving batch before the completion of the previous one. The objective is to find a schedule that minimizes the maximum lateness $L_{\max} = \max_{1 \leqslant j \leqslant n} (C_j - d_j)$ where C_j denotes the time at which job j is completed.

Our interest in the $P, NC_{inc}|r_j, q_j|C_{\max}$ was raised because it offers a unified framework for modeling a large family of parallel machine problems including, among others, two important special cases namely $P|r_j, q_j|C_{\max}$ and $P, NC_{inc}||C_{\max}$. The $P|r_j, q_j|C_{\max}$ is an important (strongly $\mathcal{NP}$-hard) scheduling problem which arises as a strong relaxation of the Multiprocessor Flow Shop problem [10,21,25]. Moreover, it plays a central role in some exact algorithms for the Resource Constrained Project Scheduling Problem [4]. Despite its theoretical and practical interest, the $P|r_j, q_j|C_{\max}$ received scant attention in the scheduling literature. In particular, Carlier [3] proposed the first branch-and-bound algorithm for this problem. Gharbi and Haouari [7] improved his algorithm by including new tools such as a preprocessing algorithm and Jackson's Pseudo Preemptive Schedule [6]. On the other hand, Lee et al. [17] investigated $P, NC_{inc}||C_{\max}$ and showed that the *Longest Processing Time* (*LPT*) rule yields a worst-case ratio of $\frac{3}{2} - \frac{1}{2m'}$, where $m' \leqslant m$. Also, Lee [16] showed that if *LPT* is appropriately modified, then it yields a worst-case ratio of $\frac{4}{3}$. Kellerer [12] developed a dual approximation heuristic which worst-case ratio is $\frac{5}{4}$. A strong lower bound for the $P, NC_{inc}||C_{\max}$ has been proposed by Webster [26].

In this paper, we present an exact branch-and-bound algorithm for solving the $P, NC_{inc}|r_j, q_j|C_{\max}$. The proposed approach is based on four main features:

(i) a strong lower bound that is based on max-flow computations
(ii) an effective heuristic that is based on heads and tails adjustments and feasibility tests

(iii) a representation of a schedule (on parallel machines) as a permutation of jobs
(iv) dominance rules that aim at reducing the size of the search tree.

In order to provide evidence of the versatility and practical usefulness of the proposed approach, we used it for solving the special cases $P, NC_{inc}||C_{\max}$ and $P|r_j, q_j|C_{\max}$, for which the approach was found to be very effective. For instance, whereas the best existing algorithm for solving the $P|r_j, q_j|C_{\max}$ experiences difficulties in solving hard instances with more than 300 jobs and 4 machines [7], the proposed algorithm makes it possible to solve instances with up to 1000 jobs and 10 machines within moderate CPU time.

The paper is organized as follows. In Section 2, we introduce some results that will be used throughout the paper. Sections 3 and 4 are devoted to the description of new lower and upper bounds for the $P, NC_{inc}|r_j, q_j|C_{\max}$. The details of our branch-and-bound algorithm are provided in Section 5. In Section 6, the performance of our algorithm is analyzed through an extensive computational study.

2. Preliminary results

For the sake of simplicity, we assume that the machines are indexed in the nondecreasing order of their availability times (i.e. $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_m$). The values $\bar{r}_k(J)$, $\bar{p}_k(J)$ and $\bar{q}_k(J)$ denote the kth smallest release date, processing time, and delivery time in J, respectively. Note that since a job j cannot start processing before $\max(r_j, a_1)$, then we implicitly assume w.n.l.g. that r_j is adjusted to $\max(r_j, a_1)$ for all $j \in J$. Similarly, the availability time of a machine M_i $(i = 1, \ldots, m)$ is adjusted to $\max(a_i, \bar{r}_1(J))$.

2.1. Bounds on the number of processing machines

It is worth noting that, due to non uniform availability times, some machines may not process any job in any optimal solution. The following observation provides simple bounds on the number of processing machines in an optimal solution.

Proposition 1. *Let UB denote an upper bound on the optimal makespan. Define*:

- $\underline{m} = \left\lceil \frac{\sum_{j\in J} p_j}{UB - a_1 - \bar{q}_1(J)} \right\rceil$
- $\bar{m}$ *as the smallest* k $(k = 1, \ldots, m-1)$ *satisfying* $a_{k+1} + \min_{j\in J}(p_j + q_j) > UB$ (*if no* k *satisfies this condition, then* $\bar{m} = m$).

Then, the number of machines m^* *that are processing in an optimal schedule satisfies* $\underline{m} \leqslant m^* \leqslant \bar{m}$.

Proof. A simple lower bound on the optimal makespan if exactly k machines are active is $LB_k = a_1 + (1/k)\sum_{j\in J} p_j + \bar{q}_1(J)$. Clearly, if $LB_k > UB$, then $m^* \geqslant k+1$. Therefore, in an optimal schedule, $m^* \geqslant \lceil \sum_{j\in J} p_j/(UB - a_1 - \bar{q}_1(J)) \rceil$.

Also, if $a_{k+1} + \min_{j\in J}(p_j + q_j) > UB$, then machine M_{k+1} cannot be used. Therefore, $m^* \leqslant k$. □

It is worth noting that Lee et al. [17] proposed an upper bound on the number of processing machines in an optimal schedule for the $P, NC_{inc}||C_{\max}$. This bound is computed by taking the smallest value of k satisfying $a_{k+1} > (\sum_{h=1}^{k} a_h + \sum_{j\in J} p_j)/k$. It is easy to check that there is no dominance relation between the latter bound and $\bar{m}$ for the $P, NC_{inc}||C_{\max}$.

Let S_i $(i=1,\ldots,\bar{m}-1)$ denote the subset of jobs that have to be processed on machines $M_1,\ldots,M_i$ in an optimal schedule. Since each job j can be processed by machine M_1, then we have

$$S_1 = \{j \in J : a_2 + p_j + q_j > UB\}$$

and

$$S_i = S_{i-1} \cup \{j \in J : a_i + p_j + q_j \leqslant UB \text{ and } a_{i+1} + p_j + q_j > UB\}$$
$$\forall i = 2,\ldots,\bar{m}-1.$$

Let i_0 denote the smallest i $(i=1,\ldots,\bar{m}-1)$ such that $S_i \neq \emptyset$. Since each machine M_h $(h=i+1,\ldots,k)$ has to process at least one job, then the following result immediately holds.

Lemma 1. *If k machines are processing in an optimal solution then we have*

$$k - i \leqslant |J \backslash S_i| \quad \forall i = i_0,\ldots,k-1.$$

Corollary 1. *Let k_0 $(k_0 = i_0+1,\ldots,\bar{m})$ denote the smallest k such that for a given i $(i=i_0,\ldots,k-1)$ we have $k - i > |J\backslash S_i|$. Then, the number of machines m^* that are processing in an optimal solution satisfies $m^* \leqslant k_0 - 1$.*

An immediate consequence of the above corollary is that the value of $\bar{m}$ is adjusted to $k_0 - 1$ (whenever k_0 exists).

2.2. Adjustments and feasibility tests

During the last few years, several authors implemented various adjustment procedures for scheduling problems [1,2,5,7,15,18,20]. In this section, we describe the so-called *Feasibility and Adjustment Procedure* (*FAP*) proposed by Gharbi and Haouari [7] for the $P|r_j, q_j|C_{\max}$. The objective of the *FAP* is twofold. It aims at adjusting the heads and tails, and checking the feasibility of a nonpreemptive schedule. The *FAP* can be extended to deal with the $P, NC_{inc}|r_j, q_j|C_{\max}$ and was found very effective in the computation of both lower and upper bounds. In order to make the paper self-contained, we briefly describe the *FAP*. For a more detailed description, the reader is referred to [7].

First, let LB and UB denote a lower and an upper bound on the optimal solution of a $P, NC_{inc}|r_j, q_j|C_{\max}$ instance. The problem is to check the feasibility of a nonpreemptive schedule with makespan less than or equal to a value $C \in [LB, UB-1]$. For that purpose, a deadline $d_j = C - q_j$ is associated with each job $j \in J$. Clearly, a schedule has a makespan less than or equal to C if and only if each job finishes processing no later than its corresponding deadline.

The *FAP* is based on the observation that if a job j is such that $d_j - r_j < 2p_j$, then in any nonpreemptive schedule, there is necessarily one machine which has to process job j during the interval $[d_j - p_j, r_j + p_j]$. Thus, a lower bound on the number of machines which are necessarily loaded at any time can be easily computed (note that a machine which is not yet available is considered as loaded). The following feasibility condition immediately holds:

Condition 1. The instance is infeasible if there is a time $t \in [a_1, \max_{j\in J} d_j]$ such that the number of machines loaded at t is strictly greater than m.

Moreover, each job $j \in S = \{j \in J; d_j - r_j < 2p_j\}$, has a *fixed* processing part of $2p_j - (d_j - r_j)$ units which has to be processed in $[d_j - p_j, r_j + p_j]$, and each job $j \in J$ has a *free* processing part of $p'_j = \min(p_j, d_j - r_j - p_j)$ units which has to be processed in $[r_j, d_j - p_j] \cup [r_j + p_j, d_j]$. Let $e_1, e_2, \ldots, e_K$ be the different values of r_j $(j \in J)$, d_j $(j \in J)$, $d_j - p_j$ $(j \in S)$, $r_j + p_j$ $(j \in S)$ and a_i $(i = 1, \ldots, m)$ ranked in increasing order. For each time interval $I_k = [e_k, e_{k+1}]$ $(k = 1, \ldots, K-1)$, we denote by J_k the set of jobs which free parts may be processed during I_k, by n_k the number of jobs in J_k, and by m_k the number of machines which are idle during I_k. Since the amount of work in I_k $(k = 1, \ldots, K-1)$ cannot exceed $A_k = \min \sum_{j\in J_k} p'_j;\ (e_{k+1} - e_k) \times \min(n_k, m_k)\}$, then the following feasibility condition holds:

Condition 2. The instance is infeasible if $\sum_{k=1}^{K-1} A_k < \sum_{j\in J} p'_j$.

Provided the number of loaded machines at any time, one can easily compute the time windows in which there is at least one idle machine. These time windows are used in order to adjust heads and tails of any job $j_0 \in J$. Indeed, a job j_0 can start processing at r_{j_0} in a feasible nonpreemptive schedule if there exists a time window $[a, b]$ such that $[r_{j_0}, r_{j_0} + p_{j_0}] \subseteq [a, b]$. Otherwise, the earliest starting time of j_0 is at least equal to the lower bound of the first time window which fits $[r_{j_0}, r_{j_0} + p_{j_0}]$. Similarly, the deadline of job j_0 can be adjusted to the upper bound of the last time window which fits $[d_{j_0} - p_{j_0}, d_{j_0}]$. After performing these adjustments, the following feasibility condition immediately holds:

Condition 3. If there is a job $j_0 \in J$ such that $r_{j_0} + p_{j_0} > d_{j_0}$, then the instance is infeasible.

The process is continued until there is no possible adjustment or an infeasibility is detected.

3. Lower bounds

In this section, we develop several new lower bounds for the $P, NC_{inc}|r_j, q_j|C_{\max}$.

3.1. Simple lower bounds

A trivial lower bound which can be computed in $O(n)$ is

$$LB_0(J) = \max_{j\in J} (r_j + p_j + q_j).$$

The following lemma provides a lower bound which takes into account the machine availability times.

Lemma 2. *A valid lower bound for the* $P, NC_{inc}|r_j, q_j|C_{\max}$ *is*

$$LB_1(J) = \min_{\underline{m} \leqslant k \leqslant \bar{m}} LB_1^k(J)$$

where $LB_1^k(J) = \left\lceil \frac{1}{k}\left(\sum_{i=1}^{k} \max(a_i, \bar{r}_i(J)) + \sum_{j\in J} p_j + \sum_{i=1}^{k} \bar{q}_i(J)\right)\right\rceil$.

Proof. It suffices to prove that LB_1^k is a valid lower bound for the k-machine instance. Denote by P_i and T_i $(i = 1, \ldots, k)$ the total processing time and the total idle time (including the unavailability time) of machine M_i in an optimal schedule, respectively. Since we have

$$P_i + T_i \leqslant C^*_{\max} \quad \text{for all } i = 1, \ldots, k$$

then

$$\left\lceil \frac{1}{k}\left(\sum_{j\in J} p_j + \sum_{i=1}^{k} T_i\right)\right\rceil \leqslant C^*_{\max}. \tag{1}$$

Note that each machine M_i has to wait an amount of time W_i before starting the processing of the jobs. Moreover, there is necessarily a first machine that has to be idle from time $C^*_{\max} - \overline{q}_1(J)$ to $C^*_{\max}$, a second one from time $C^*_{\max} - \overline{q}_2(J)$ to $C^*_{\max} \ldots$, and a kth one from time $C^*_{\max} - \overline{q}_k(J)$ to $C^*_{\max}$. Thus, we have

$$\sum_{i=1}^{k} W_i + \sum_{i=1}^{k} \bar{q}_i(J) \leqslant \sum_{i=1}^{k} T_i. \tag{2}$$

Now, we prove that a valid lower bound on $\sum_{i=1}^{k} W_i$ is $\sum_{i=1}^{k} \max(a_i, \bar{r}_i(J))$. Note that each machine M_i $(i = 1, \ldots, k)$ cannot start processing before time $\max(a_i, r_i)$ where r_i denotes the release date of the first job to be processed on M_i. Clearly, if the objective is to minimize $\sum_{i=1}^{k} W_i$ then there is an optimal schedule σ such that $r_i \in \{\bar{r}_1(J), \bar{r}_2(J), \ldots, \bar{r}_k(J)\}$ for all $i = 1, \ldots, k$. Assume that machine M_1 starts at $\max(a_1, \bar{r}_h(J))$ in σ $(1 < h \leqslant k)$. That is, there is a machine M_l $(1 < l \leqslant k)$ that starts at $\max(a_l, \bar{r}_1(J))$. Consider the schedule σ' obtained by interchanging the set of jobs assigned to machines M_1 and M_l in σ. Let W'_i denote the machine waiting time of machine M_i in σ'. We have

$$\begin{aligned}
\Delta &= \sum_{i=1}^{k} W'_i - \sum_{i=1}^{k} W_i \\
&= W'_1 + W'_l - W_1 - W_l \\
&= \max(a_1, \bar{r}_1(J)) + \max(a_l, \bar{r}_h(J)) - \max(a_1, \bar{r}_h(J)) - \max(a_l, \bar{r}_1(J)).
\end{aligned}$$

Three cases are considered:

(i) $a_l \leqslant \bar{r}_1(J)$: then $\Delta = 0$,

(ii) $\bar{r}_1(J) \leqslant a_l \leqslant \bar{r}_h(J)$: then $\Delta = \max(a_1, \bar{r}_1(J)) - a_l \leqslant 0$,
(iii) $\bar{r}_h(J) \leqslant a_l$: then $\Delta = \max(a_1, \bar{r}_1(J)) - \max(a_1, \bar{r}_h(J)) \leqslant 0$.

Therefore $\sum_{i=1}^{k} W'_i \leqslant \sum_{i=1}^{k} W_i$. Since σ is optimal, then σ' is an optimal schedule where machine M_1 starts processing at $\max(a_1, \bar{r}_1(J))$. Similarly, it is possible to interchange jobs in σ' in order to obtain an optimal schedule where machines M_1 and M_2 start processing at $\max(a_1, \bar{r}_1(J))$ and $\max(a_2, \bar{r}_2(J))$, respectively, and so on. Therefore,

$$\sum_{i=1}^{k} \max(a_i, \bar{r}_i(J)) \leqslant \sum_{i=1}^{k} W_i. \tag{3}$$

Combining (1)–(3) completes the proof. □

Note that, given LB_1^k, the value of LB_1^{k-1} can be computed in O(1). Since $LB_1^{\bar{m}}$ can be computed in $O(n \log m)$, then the computation of LB_1 requires $O(n \log m + m)$ time.

3.2. A subset-sum based lower bound

Assume that exactly k machines are active in an optimal schedule ($\underline{m} \leqslant k \leqslant \bar{m}$). Let $J_{i,k}$ $(i = i_0, \ldots, k-1)$ denote the subset of jobs that are processed on the machine subset $\{M_1, \ldots, M_i\}$. We have

$$\frac{1}{i}\left(\sum_{h=1}^{i} \max(a_h, \bar{r}_h(J_{i,k})) + \sum_{j \in J_{i,k}} p_j + \sum_{h=1}^{i} \bar{q}_h(J_{i,k})\right) \leqslant C_{\max}.$$

Thus, a valid lower bound is

$$L^1_{i,k}(J_{i,k}) = \frac{1}{i}\left(\sum_{h=1}^{i} \max(a_h, \bar{r}_h(J)) + \sum_{j \in J_{i,k}} p_j + \sum_{h=1}^{i} \bar{q}_h(J)\right).$$

Obviously, since the jobs of $J \backslash J_{i,k}$ are processed on the machine subset $\{M_{i+1}, \ldots, M_k\}$, then we have

$$\frac{1}{k-i}\left(\sum_{h=i+1}^{k} \max(a_h, \bar{r}_{h-i}(J \backslash J_{i,k})) + \sum_{j \in J \backslash J_{i,k}} p_j + \sum_{h=1}^{k-i} \bar{q}_h(J \backslash J_{i,k})\right) \leqslant C_{\max}.$$

Note that $S_i \subseteq J_{i,k}$ (where S_i is the set defined in Section 2.1). Then, a second valid lower bound is

$$L^2_{i,k}(J_{i,k}) = \frac{1}{k-i}\left(\sum_{h=i+1}^{k} \max(a_h, \bar{r}_{h-i}(J \backslash S_i)) + \sum_{j \in J \backslash J_{i,k}} p_j + \sum_{h=1}^{k-i} \bar{q}_h(J \backslash S_i)\right).$$

Hence, for a given subset $J_{i,k} \subseteq J$, a valid lower bound is

$$L_{i,k}(J_{i,k}) = \max\{L^1_{i,k}(J_{i,k}), L^2_{i,k}(J_{i,k})\}.$$

Define:

$$LB_{i,k}(J) = \min_{J_{i,k}\subseteq J} L_{i,k}(J_{i,k}) \quad \forall i_0 \leqslant i \leqslant k-1; \;\; \underline{m} \leqslant k \leqslant \bar{m},$$

$$LB_k(J) = \max_{i_0 \leqslant i \leqslant k-1} LB_{i,k}(J) \quad \forall \underline{m} \leqslant k \leqslant \bar{m}.$$

Then, a valid lower bound is

$$LB_2(J) = \min_{\underline{m} \leqslant k \leqslant \bar{m}} LB_k(J).$$

Now, we show how to compute $LB_{i,k}(J)$. For given values of i and k, define the vector $y \in \{0,1\}^n$ in the following way:

$$y_j = \begin{cases} 1 & \text{if } j \in J_{i,k} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } j \in J.$$

Since, $y_j = 1$ for all $j \in S_i$, then we have

$$L^1_{i,k}(J_{i,k}) = \frac{1}{i}\left(\sum_{h=1}^{i} \max(a_h, \bar{r}_h(J)) + \sum_{j\in S_i} p_j + \sum_{j\in J\setminus S_i} p_j y_j + \sum_{h=1}^{i} \bar{q}_h(J)\right),$$

$$L^2_{i,k}(J_{i,k}) = \frac{1}{k-i}\left(\sum_{h=i+1}^{k} \max(a_h, \bar{r}_{h-i}(J\setminus S_i)) + \sum_{j\in J\setminus S_i} p_j - \sum_{j\in J\setminus S_i} p_j y_j + \sum_{h=1}^{k-i} \bar{q}_h(J\setminus S_i)\right)$$

Define A_i, $B_{i,k}$ and $f_{i,k}(J,y)$ by

$$A_i = \sum_{h=1}^{i} \max(a_h, \bar{r}_h(J)) + \sum_{j\in S_i} p_j + \sum_{h=1}^{i} \bar{q}_h(J),$$

$$B_{i,k} = \sum_{h=i+1}^{k} \max(a_h, \bar{r}_{h-i}(J\setminus S_i)) + \sum_{j\in J\setminus S_i} p_j + \sum_{h=1}^{k-i} \bar{q}_h(J\setminus S_i),$$

$$f_{i,k}(J,y) = \max\left\{\frac{A_i + \sum_{j\in J\setminus S_i} p_j y_j}{i}, \frac{B_{i,k} - \sum_{j\in J\setminus S_i} p_j y_j}{k-i}\right\}.$$

We have $LB_{i,k}(J) = \min_y f_{i,k}(J,y)$. Note that $f_{i,k}(J,y) = (A_i + \sum_{j\in J\setminus S_i} p_j y_j)/i$ if and only if $\sum_{j\in J\setminus S_i} p_j y_j \geqslant b_{ik} = (i B_{i,k} - (k-i)A_i)/k$. Therefore $LB_{i,k}(J)$ can be computed by solving the pair of the following *Subset-Sum* problems:

$$z_1 = Min \; \frac{A_i + \sum_{j\in J\setminus S_i} p_j y_j}{i}$$

subject to :

$$\sum_{j\in J\setminus S_i} p_j y_j \geqslant b_{ik}$$

$$y_j \in \{0,1\} j \in J\setminus S_i$$

and

$$z_2 = Min\ \frac{B_{i,k} - \sum_{j\in J\setminus S_i} p_j y_j}{k-i}$$

subject to :

$$\sum_{j\in J\setminus S_i} p_j y_j \leqslant b_{ik}$$
$$y_j \in \{0,1\} j \in J\setminus S_i.$$

We have $LB_{i,k}(J) = \min(z_1, z_2)$.

Hence, the computation of $LB_{i,k}(J)$ requires the *exact* solution of a Subset-Sum problem (SSP). This problem is known to be $\mathcal{NP}$ -hard [19]. Nevertheless, during the last few years, several high performance exact and approximate algorithms have been proposed for this problem [13,22,24]. However, since the problem is $\mathcal{NP}$-hard, one would reasonably expect that for some instances it might take an excessive computation time to get a proven optimal solution. Therefore, in the sequel, we show how a significantly simpler version of $LB_{i,k}$ could be obtained. To that aim, we assume that the value of $x = \sum_{j\in J\setminus S_i} p_j y_j$ could be equal to any integer lying in $[0, \sum_{j\in J\setminus S_i} p_j]$.

Note that it is implicitly assumed that there are at least $k-i$ jobs in $J\setminus J_i$. Therefore, the total load of machines $M_{i+1}, \ldots, M_k$ is at least equal to $\sum_{j=1}^{k-i} \bar{p}_j(J\setminus S_i)$. That is, $\sum_{j\in J\setminus S_i} p_j - x \geqslant \sum_{j=1}^{k-i} \bar{p}_j(J\setminus S_i)$. Consequently, x actually lies in $[0, \sum_{j=k-i+1}^{|J\setminus S_i|} \bar{p}_j(J\setminus S_i)]$.

First, assume that $A_i/i \geqslant B_{i,k}/(k-i)$. Then $(A_i + x)/i \geqslant (B_{i,k} - x)/(k-i)$ for all $x \in [0, \sum_{j=k-i+1}^{|J\setminus S_i|} \bar{p}_j(J\setminus S_i)]$. Therefore $LB_{i,k}(J) = A_i/i$. Now, assume that $A_i/i < B_{i,k}/(k-i)$. Note that $(A_i + x)/i$ is an increasing linear function and $(B_{i,k} - x)/(k-i)$ is a decreasing linear function. Let x_0 denote the value of x such that $(A_i + x)/i = (B_{i,k} - x)/(k-i)$. That is, $x_0 = (iB_{i,k} - (k-i)A_i)/k$. Three cases have to be considered:

(i) if $0 < x_0 < \bar{p}_1(J\setminus S_i)$: then $LB_{i,k}(J) = \min\{f_{i,k}(J, 0),\ f_{i,k}(J, \bar{p}_1(J\setminus S_i))\}$.
(ii) if $\bar{p}_1(J\setminus S_i) \leqslant x_0 \leqslant \sum_{j=k-i+1}^{|J\setminus S_i|} \bar{p}_j(J\setminus S_i)$: if x_0 is integer then $LB_{i,k}(J) = f_{i,k}(J, x_0)$. Otherwise, $LB_{i,k}(J) = \min\{f_{i,k}(J, \lfloor x_0 \rfloor),\ f_{i,k}(J, \lceil x_0 \rceil)\}$.
(iii) if $x_0 > \sum_{j=k-i+1}^{|J\setminus S_i|} \bar{p}_j(J\setminus S_i)$: then $LB_{i,k}(J) = f_{i,k}(J, \sum_{j=k-i+1}^{|J\setminus S_i|} \bar{p}_j(J\setminus S_i))$.

Note that it is assumed that $i_0 < k$ in the computation of $LB_{i,k}(J)$. For values of k such that $k \leqslant i_0$, the lower bound $LB_k(J)$ can be replaced by $LB_1^k(J)$. Clearly, we have

$$LB_2(J) \geqslant LB_1(J).$$

3.3. A max-flow-based lower bound

The lower bound introduced in this section consists in repeatedly checking the existence of a relaxed schedule with makespan less than or equal to a trial value C. For that purpose, a first step consists in applying *FAP* to the $P, NC_{inc}|r_j, q_j, d_j|C_{\max}$ defined by associating with each job $j \in J$ a deadline $d_j = C - q_j$. Secondly, the feasibility of the trial value C is checked using a max-flow formulation as follows. A *semi-preemptive* schedule is defined as a schedule where the fixed parts of the jobs are constrained to start and to finish at fixed times with no preemption, whereas the free parts can be preempted. This concept of

semi-preemptive scheduling was recently introduced by Haouari and Gharbi [9] and used to derive a max-flow-based lower bound for the $P|r_j, q_j|C_{\max}$ which dominates the classical preemptive lower bound [11]. In this section, we extend this work in a non trivial way and we show how a tight semi-preemptive lower bound can be derived for the $P, NC_{inc}|r_j, q_j|C_{\max}$.

For each job j, let μ_j denote the largest machine index such that $a_i + p_j \leqslant d_j$. Clearly, in any nonpreemptive schedule a job j cannot be scheduled on any machine M_i such that $i > \mu_j$. Now, we show how to check the feasibility of a semi-preemptive schedule with the two additional conditions:

$\mathbf{C}_1$: Each machine M_i $(1 \leqslant i \leqslant m)$ is constrained to start processing after its availability time a_i.
$\mathbf{C}_2$: Each job $j \in J$ can only be scheduled on a machine M_i such that $1 \leqslant i \leqslant \mu_j$.

According to the notation of Section 2.2, we assume w.n.l.g. that the m_k machines which are idle during the time interval $I_k = [e_k, e_{k+1}]$ $(1 \leqslant k \leqslant K-1)$ are indexed $1, 2, \ldots, m_k$, respectively. The feasibility problem can be solved using the following extension of Horn's approach [11]:

Consider the network $N = (V, A)$ where the set of nodes V is the union of the following subsets:

- Job nodes $\{J_1, \ldots, J_n\}$.
- Time interval nodes $E_k = \{E_k^1, E_k^2, \ldots, E_k^{m_k}\}$ $(1 \leqslant k \leqslant K-1)$. Each time interval node E_k^h $(h = 1, \ldots, m_k)$ represents the time interval $I_k = [e_k, e_{k+1}]$ but with only the subset of machines $\{M_1, M_2, \ldots, M_h\}$ being available.
- $\{s, t\}$ where s is the source node, and t is the sink node

The set of arcs A is constructed in the following way:

- For each job node J_j $(j = 1, \ldots, n)$ such that $p'_j > 0$, there is an arc (s, J_j) with capacity p'_j representing the free part of job j.
- For each $k = 1, \ldots, K-1$ and $h = 1, \ldots, m_k - 1$, there is an arc (E_k^h, E_k^{h+1}) with capacity $h(e_{k+1} - e_k)$.
- For each $k = 1, \ldots, K-1$, there is an arc $(E_k^{m_k}, t)$ with capacity $m_k(e_{k+1} - e_k)$.
- For each $j = 1, \ldots, n$, $k = 1, \ldots, K-1$, and $h = 1, \ldots, m_k$, there is an arc (J_j, E_k^h) with capacity $e_{k+1} - e_k$ if and only if $h = \min(\mu_j, m_k)$ and one of the three following conditions holds:

  (i) $d_j - r_j < 2p_j$, $r_j \leqslant e_k$ and $e_{k+1} \leqslant d_j - p_j$.
  (ii) $d_j - r_j < 2p_j$, $r_j + p_j \leqslant e_k$ and $e_{k+1} \leqslant d_j$.
  (iii) $d_j - r_j \geqslant 2p_j$, $r_j \leqslant e_k$ and $e_{k+1} \leqslant d_j$.

Obviously, an interval node which is not connected with any job node is dropped from the network. We have the following theorem.

Theorem 1. *A semi-preemptive schedule respecting time windows and conditions* (C_1) *and* (C_2) *exists if and only if the maximum flow between nodes s and t is equal to* $\sum_{j \in J} p'_j$.

Proof. Firstly, assume that a semi-preemptive schedule respecting conditions (C_1) and (C_2) exists. For each k ($k = 1, \ldots, K-1$), define τ_{jk}^i as the total processing time of job j ($j = 1, \ldots, n$) on machine M_i ($i = 1, \ldots, m$) in the time interval I_k. A corresponding flow (w, x, y, z) is obtained in the following way:

(i) A flow $w_j = p'_j$ is assigned to each arc $(s, J_j) \in A$.

(ii) A flow $x_{jk}^h = \sum_{i=1}^{h} \tau_{jk}^i$ is assigned to each arc $(J_j, E_k^h) \in A$ where $h = \min(\mu_j, m_k)$. The flow variable x_{jk}^h is equal to the total processing time of job j on machines $M_1, \ldots, M_h$ in the time interval I_k. Then, it satisfies $x_{jk}^h \leqslant e_{k+1} - e_k$.

(iii) A flow $y_k^h = y_k^{h-1} + \sum_{j=1}^{n} x_{jk}^h$ (with $y_k^0 = 0$) is assigned to each arc $(E_k^h, E_k^{h+1}) \in A$. The flow variable y_k^h is equal to the cumulative load of machines $M_1, \ldots, M_h$ in the time interval I_k. Then, it satisfies $y_k^h \leqslant h(e_{k+1} - e_k)$.

(iv) A flow $z_k = y_k^{m_k - 1} + \sum_{j=1}^{n} x_{jk}^{m_k}$ is assigned to each arc $(E_k^{m_k}, t) \in A$. This flow can also be expressed as $z_k = \sum_{h=1}^{m_k} \sum_{j=1}^{n} x_{jk}^h$. Therefore, z_k is equal to the total load of all the machines that are available during the time interval I_k. Consequently, $z_k \leqslant m_k(e_{k+1} - e_k)$.

Moreover, since the schedule is feasible, then the total processing time of job j is equal to its free processing part. Hence, $\sum_{k=1}^{K} \sum_{h=1}^{m_k} x_{jk}^h = p'_j$. Therefore, the flow (w, x, y, z) satisfies both the capacity and the flow conservation constraints. Hence, it is feasible. This flow is maximal because its value is $\sum_{j=1}^{n} p'_j$ which is equal to the capacity of the cutset $(\{s\} : V \backslash \{s\})$.

Conversely, given a feasible flow (w, x, y, z) with value $\sum_{j=1}^{n} p'_j$, a feasible semi-preemptive schedule could be constructed in the following way. Firstly, we assign the fixed parts by successively loading machines M_m, M_{m-1}, etc. Secondly, for each time interval I_k, we schedule jobs satisfying $\sum_{h=1}^{m_k} x_{jk}^h > 0$ on m_k identical machines, while satisfying the additional condition (C_2). A schedule meeting this condition is constructed by scheduling the jobs according to nondecreasing μ_j on the first available machine and splitting jobs into two parts whenever the upper bound e_{k+1} is met. The remaining part of the job is scheduled on the next machine at time e_k.

It is easy to check that the resulting schedule is necessarily feasible. Indeed, assume that for some time interval I_k we schedule jobs $J_1, J_2, \ldots, J_{p-1}$ but we fail to schedule job J_p (with $\mu_p = h^*$) on any machine $M_1, M_2, \ldots, M_{h^*}$. Therefore, we have

$$\sum_{j=1}^{p} \sum_{h=1}^{h^*} x_{jk}^h > h^*(e_{k+1} - e_k). \tag{4}$$

However, since the flow is feasible then $y_k^{h^*} \leqslant h^*(e_{k+1} - e_k)$. The value of the flow $y_k^{h^*}$ is equal to $y_k^{h^*} = y_k^{h^*-1} + \sum_{j=1}^{n} x_{jk}^{h^*} = \sum_{j=1}^{n} \sum_{h=1}^{h^*} x_{jk}^h$. Therefore

$$\sum_{j=1}^{p} \sum_{h=1}^{h^*} x_{jk}^h \leqslant \sum_{j=1}^{n} \sum_{h=1}^{h^*} x_{jk}^h \leqslant h^*(e_{k+1} - e_k), \tag{5}$$

which contradicts (4). Hence, the resulting schedule is semi-preemptive, satisfies conditions (C_1)–(C_2), and time windows. □

Table 1
Data of the 4 job—3 machine instance of Example 1

j	1	2	3	4
r_j	3	3	4	4
p_j	5	4	6	1
d_j	12	12	15	12

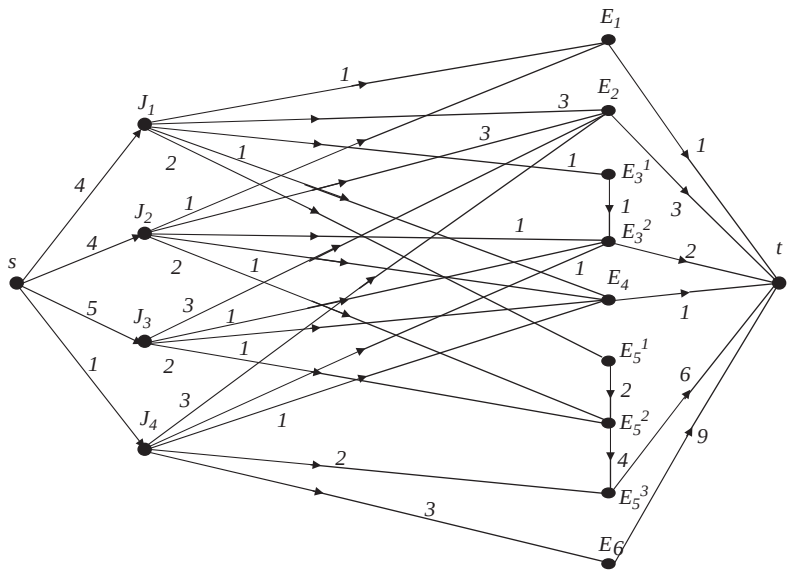

Fig. 1. The flow network corresponding to Example 1.

The computation of the maximum flow requires $O(N^3)$ time, where N is the number of nodes in the network. We have a maximum of $4nm + m^2 - m + n + 2$ nodes. Thus, after applying *FAP*, checking the existence of a semi-preemptive schedule with $C_{\max}$ less than or equal to C requires $O(n^3m^3)$ time.

If LB and UB denote a lower and upper bound on the optimal makespan, respectively, then the optimal semi-preemptive schedule is computed using a bisection search on the interval $[LB, UB]$. The obtained lower bound, denoted by $LB_3(J)$, can be computed in $O(n^3m^3(\log n + \log m + \log p_{\max}))$ [14].

Example 1. Consider the feasibility problem defined on the 4 job—3 machine instance which data are depicted in Table 1 .

Assume that the machine availabilities are $a_1 = 3$, $a_2 = 8$ and $a_3 = 10$. Then, we have

$$\mu_1 = 1, \quad \mu_2 = 2, \quad \mu_3 = 2, \quad \mu_4 = 3.$$

The free parts of the jobs are $p'_1 = 4$, $p'_2 = 4$, $p'_3 = 5$ and $p'_4 = 1$. The time intervals corresponding to this problem are $E_1 = [3, 4]$, $E_2 = [4, 7]$, $E_3 = [8, 9]$, $E_4 = [9, 10]$, $E_5 = [10, 12]$ and $E_6 = [12, 15]$. Their respective number of available machines are 1, 1, 2, 1, 3 and 3. The flow network corresponding to the semi-preemptive lower bound is illustrated in Fig. 1.

4. Upper bounds

4.1. Jackson's schedule

Jackson's schedule provides a simple approximation of the $P, NC_{inc}|r_j, q_j|C_{\max}$ optimal solution. This algorithm is based on a dispatching rule which schedules the available job with the largest tail on the first available machine [3]. The makespan of Jackson's schedule will be denoted by $JS(J)$. Its computation requires $O(n \log n)$ time.

4.2. FAP-based upper bounds

Although Jackson's schedule is a fairly good approximation schedule, its shortsightedness constitutes its major flaw. In this section we describe how we can use the *FAP* in order to anticipate at best the impact of the current decision. Assume that we are interested in constructing a nonpreemptive schedule with makespan less than or equal to a trial value $C \in [LB, UB - 1]$. First, we set $d_j = C - q_j$ for all $j \in J$ and we adjust the heads and the tails using *FAP*. A job $j \in J$ such that $d_j = r_j + p_j$ is referred to as a fixed job and is considered as already scheduled. Let L denote the list of the free (unscheduled) jobs in J sorted according to the nondecreasing order of their heads, where ties are settled according to the nonincreasing order of tails. At each iteration, we use *FAP* to check whether the first job $j_0 \in L$ can be scheduled at its release date (note that after applying *FAP*, all the release dates are larger than the smallest machine availability). In this case, we set $d_{j_0} = r_{j_0} + p_{j_0}$. The list L is then updated by the *FAP*.

Now, assume that the *FAP* proves that scheduling j_0 at the current position yields an infeasibility. Therefore, we skip job j_0 and move to the next job in the list. Note that there may be no possible job to be scheduled at the current iteration. In this case, we update the trial value to $C + 1$ and so on. The algorithm stops when a feasible schedule is constructed. In the sequel, the makespan of this approximate schedule will be denoted by $FAP\_UB(J)$.

A second variant of the *FAP*-based upper bound amounts to a backward construction of the schedule (i.e. starting from the last scheduled job). Now, the list L contains the free jobs in J sorted according to the nondecreasing order of their tails, where ties are settled according to the nonincreasing order of heads. Also, for a potential job j_0 to be scheduled, we set $r_{j_0} = d_{j_0} - p_{j_0}$. The obtained approximate makespan is denoted by $FAP\_UB^{-1}(J)$. We found in our experiments that taking the best of the two obtained schedules often yields an accurate approximate schedule.

5. Description of the B&B algorithm

5.1. Data representation

In our branch-and-bound algorithm, a schedule is represented by the chronological order of the jobs. Formally, with a given schedule $\sigma = (t_1, t_2, \ldots, t_n)$ is associated a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ such that $t_{\pi(k)} \leqslant t_{\pi(k+1)}$ for all $k = 1, \ldots, n-1$. It is worth noting that Carlier and Néron [5] used a similar representation for solving the Hybrid Flow Shop problem. Starting at the root node with an empty permutation, at each level of the tree, a job is assigned to the first available position of the permutation. With each node N of the search tree, we associate a partial permutation π_N of scheduled jobs. It means that node N represents all the permutations beginning with π_N. Let $\bar{J}(N)$ denote the set of unscheduled jobs. We assume w.n.l.g. that $a_1(N) \leqslant a_2(N) \leqslant \cdots \leqslant a_m(N)$.

A lower bound $LB(N)$, an upper bound $UB(N)$ and deadlines $d_j(N)$ are associated with node N. If UB denotes the current best upper bound, then the deadlines are set to $d_j(N) = UB - q_j(N) - 1$ for $1 \leqslant j \leqslant n$. Moreover, a starting time $t_j(N)$ is defined for all $j \in J$.

5.2. Branching rule

The purpose of the branching rule is to indicate a candidate job to be fixed on the first available position of the partial permutation π_N associated with a given node N, and to generate a descendant of the current node in the search tree. Given a node N_0, if a job $j_0 \in \bar{J}(N_0)$ is appended to π_{N_0}, then a descendant node N of N_0 is created with the following data:

Machine data

- $a_i(N) = a_i(N_0)$ for $i = 1, \ldots, m$,
- $a_1(N) = r_{j_0}(N_0) + p_{j_0}$,
- Update $a_i(N)$ for $i = 1, \ldots, m$.

Data of scheduled jobs

- $\pi_N = \pi_{N_0} j_0$,
- $t_{j_0}(N) = r_{j_0}(N_0)$,
- $d_{j_0}(N) = r_{j_0}(N_0) + p_{j_0}$,
- $q_{j_0}(N) = \max(q_{j_0}(N), LB(N_0) - d_{j_0}(N))$.

Data of unscheduled jobs

- $\bar{J}(N) = \bar{J}(N_0) \backslash \{j_0\}$,
- $r_j(N) = \max(r_j(N_0), t_{j_0}(N), a_1(N))$ for all $j \in \bar{J}(N)$,
- $q_j(N) = q_j(N_0)$ for all $j \in \bar{J}(N)$,
- $d_j(N) = d_j(N_0)$ for all $j \in \bar{J}(N)$.

During the computations, each time a new improved upper bound UB is found, the deadline and the tail of all the jobs $j \in \bar{J}(N)$ are adjusted to

$$d_j(N) = \min\{d_j(N), UB - q_j(N) - 1\},$$
$$q_j(N) = \max\{q_j(N), LB(N) - d_j(N)\}.$$

The depth-first search strategy has been adopted. It consists in branching the first candidate node descendant of the current node in the tree. W.n.l.g., the jobs of $\bar{J}(N)$ are ranked according to nondecreasing release dates, and in case of ties, nonincreasing delivery times and nondecreasing processing times.

For the sake of clarity, we will denote the partial permutation π_N by π, the set $\bar{J}(N)$ by $\bar{J}$, and so on.

5.3. Dominance rules

In this section, we derive immediate selection rules which aims at removing dominated nodes from the set of candidate nodes to be branched.

Let $C^*_{\max}(\pi)$ denote the minimum makespan of all those of the permutations beginning with the partial permutation π. The three following results will be used to derive dominance relations between jobs of $\bar{J}$ to be appended to π.

Observation 1. *Let j and j' be two jobs of $\bar{J}$ such that*

$$r_j = r_{j'}, \quad p_j = p_{j'} \quad \text{and} \quad q_j = q_{j'}.$$

Then,

$$C^*_{\max}(\pi j) = C^*_{\max}(\pi j').$$

Proof. Obvious. □

Observation 2. *Let $j_0 \in \bar{J}$ denote the job such that $r_{j_0} + p_{j_0} = \min_{j\in\bar{J}}(r_j + p_j)$. Assume that there exists a job $j \in \bar{J}$ such that*

$$r_j \geqslant r_{j_0} + p_{j_0}.$$

Then,

$$C^*_{\max}(\pi j_0 j) \leqslant C^*_{\max}(\pi j).$$

Proof. Consider any permutation beginning with the partial permutation πj. Sequencing job j_0 between π and j will decrease the starting time of j_0 without delaying the starting times of the other jobs. □

Observation 3. *Assume that there is a job j_0 such that $r_{j_0} \geqslant a_2$. Let j denote a job such that $r_j \leqslant r_{j_0}$. Then,*

$$C^*_{\max}(\pi j j_0) \leqslant C^*_{\max}(\pi j_0 j).$$

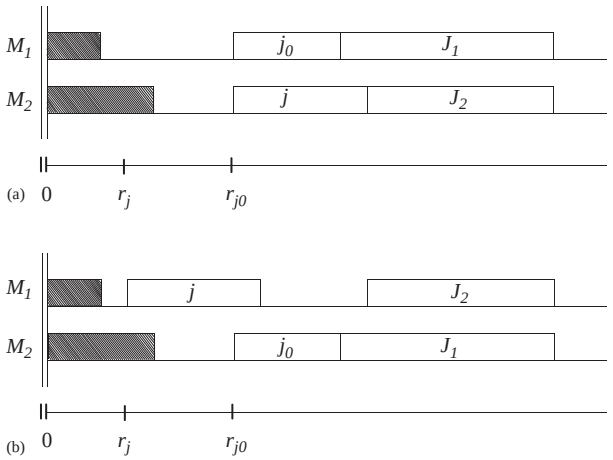

Fig. 2. $C^*_{\max}(\pi j j_0) \leqslant C^*_{\max}(\pi j_0 j)$: (a) a schedule beginning with $\pi j_0 j$, (b) a schedule beginning with $\pi j_0 j$.

Proof. Let M_1 and M_2 denote the first and second available machines. In any permutation beginning with the partial permutation $\pi j_0 j$, the job j_0 is scheduled at $t_{j_0} = r_{j_0}$ on M_1 and the job j is scheduled at $t_{j_0} = \max(r_j, t_{j_0}, a_2)$ on M_2 (see Fig. 2a). Let J_1 denote the set containing j_0 and all jobs that are processed after j_0 on M_1, and J_2 denote the set containing job j and all jobs processed after j on M_2. Since all jobs in J_1 and J_2 start processing after $t_{j_0} \geqslant \max(a_1, a_2)$, then the schedule obtained by interchanging J_1 and J_2 has the same makespan as the original one. This latter schedule is clearly dominated by the permutation beginning with $\pi j j_0$ and depicted in Fig. 2b. □

The following dominance rules are immediate consequences of the above observations. The first two rules are derived from Observations 1 and 2, respectively, whereas the two last ones are derived from Observation 3. The jobs $j_1, j_2, \ldots, j_K$ denote the jobs of $\bar{J}$ sorted according to the nondecreasing order of their release dates.

- R_1: If two jobs j_k and j_{k+1} of $\bar{J}$ have equal heads, processing times and tails, then job j_{k+1} is not candidate to be appended to π.
- R_2: All jobs $k \in \bar{J}$ such that $r_k \geqslant \min_{j \in \bar{J}} (r_j + p_j)$ are not candidate to be appended to π.
- R_3: Assume that there is a job $j_k \in \bar{J}$ such that $r_{j_k} \geqslant a_2$. Then, only jobs j_h $(h = k + 1, \ldots, K)$ are candidate to be appended to πj_k.
- R_4: If $r_{j_K} \geqslant a_2$, then job j_K is not candidate to be appended to π.

The following example shows how these selection rules are used in reducing the number of nodes in the tree.

Example 2. Consider a given partial permutation π in a two-machine instance. Let $\bar{J} = \{1, 2, 3, 4, 5\}$ which data are provided by Table 2 . Assume that the machine availabilities are $a_1 = 2$ and $a_2 = 8$.

Table 2
Data of the unscheduled jobs of Example 2

j	1	2	3	4	5
r_j	2	2	5	12	13
p_j	6	6	1	6	2
q_j	1	1	3	1	8

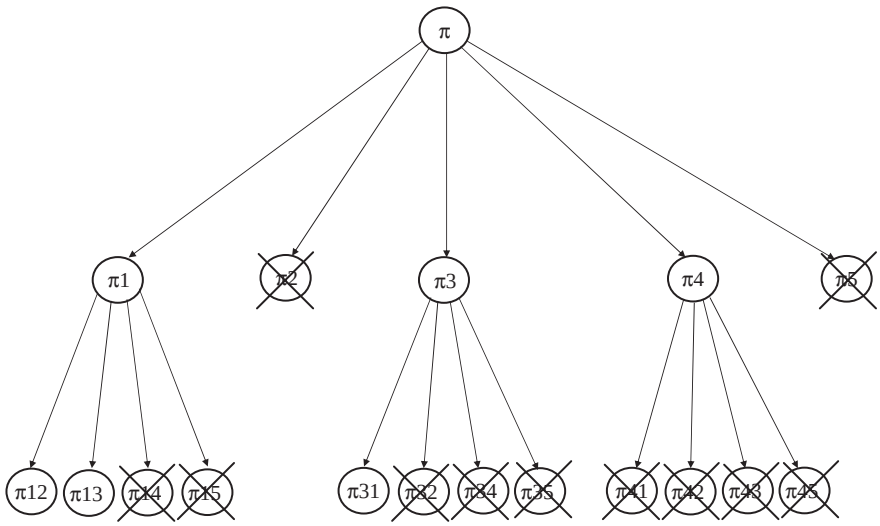

Fig. 3. Application of dominance rules.

Fig. 3 illustrates the significant impact of the proposed dominance rules on the size of the tree. Indeed, applying the dominance rules reduces the number of potentially valid nodes at the second level from 20 nodes to only 3 nodes. The details are provided in the following.

Clearly, nodes $\pi 2$ and $\pi 5$ are removed according to R_1 and R_4, respectively. Also, nodes $\pi 41$, $\pi 42$ and $\pi 43$ are removed according to R_3.

Assume that job 1 is appended to π. Then, the release dates corresponding to $\bar{J} = \{2, 3, 4, 5\}$ are $\{8, 8, 12, 13\}$. Therefore, according to R_2, the jobs with release dates larger than or equal to $\min_{j\in\bar{J}}(r_j + p_j) = 9$ are not candidate to be appended to $\pi 1$. That is, nodes $\pi 14$ and $\pi 15$ are removed.

Similarly, assume that job 3 is appended to π. Then, the release dates corresponding to $\bar{J} = \{1, 2, 4, 5\}$ are $\{6, 6, 12, 13\}$. Therefore, node $\pi 32$ is removed according to R_1, and nodes $\pi 34$ and $\pi 35$ are removed according to R_2.

Finally, in the case where job 4 is appended to π, the release dates corresponding to $\bar{J} = \{1, 2, 3, 5\}$ are $\{12, 12, 12, 13\}$. Since $r_5 = \min_{j\in\bar{J}}(r_j + p_j)$, then node $\pi 45$ is removed according to R_2.

5.4. Synthesis of the branch-and-bound algorithm

We implemented our algorithm using, at each node N, the lower bound

$$LB(N) = \max\{LB_0(J), LB_3(J)\}.$$

The upper bound computed at the root node is

$$UB(R) = \min\{JS(J), FAP\_UB(J)FAP\_UB^{-1}(J)\}.$$

It is worth noting that, since a node which yields an optimal makespan equal to UB is of no interest, then the computation of $\underline{m}$, $\overline{m}$ and S_i $(i=1,\ldots,\bar{m}-1)$ have to be slightly modified. Indeed, at each node of the tree, we have $\underline{m} = \lfloor(\sum_{j\in J} p_j)/(UB - a_1 - \bar{q}_1(J))\rfloor + 1$ and $\bar{m}$ is the smallest k $(k = 1,\ldots,m-1)$ satisfying $a_{k+1} + \min_{j\in J}(p_j + q_j) \geqslant UB$. Moreover, the subsets S_i $(i = 1,\ldots,\bar{m}-1)$ are computed by

$$S_1 = \{j \in J : a_2 + p_j + q_j \geqslant UB\}$$

and

$$S_i = S_{i-1} \cup \{j \in J : a_i + p_j + q_j < UB \text{ and } a_{i+1} + p_j + q_j \geqslant UB\}$$
$$\forall i = 2,\ldots,\bar{m}-1.$$

Note that $\underline{m}$ may be strictly larger than $\bar{m}$. In this case, the node will be pruned.

In the following pseudo-code description of our branch-and-bound algorithm, we adopted the following notation:

- N_p: the parent node of N,
- $\Omega(N)$: the set of candidate descendant nodes of N,
- N_0: the current node to be branched.

Step 0: **Initialization**

0.1. *Make a root node R containing the data set of the problem.*
0.2. *Compute UB(R) and LB(R). Set UB = UB(R).*
0.3. *If LB(R) = UB, then go to Step* 5. *Else, compute* $\Omega(R)$ *using the selection rules and set* $N_0 = R$.

Step 1: **Node selection**
If $\Omega(N_0) \neq \emptyset$, *then select a node* $N \in \Omega(N_0)$ *and go to Step* 2. *Else, go to Step* 4
Step 2: **Branching**

2.1. *Create the data of node N as described in Section* 5.3.
2.2. *Compute LB(N). If* $LB(N) \geqslant UB$ *then go to step* 3. *Else set* $q_j = \max\{q_j, LB(N) - d_j\}$ *for all* $j \in \bar{J}$.
2.3. *Apply FAP to node N*.
2.4. *Compute* $\Omega(N)$ *using the selection rules.*
2.5. *Set* $N_0 = N$ *and go to Step* 1.

Step 3: **Pruning**
Prune N and go to Step 1.
Step 4: **Backtracking**
If $N_0 = R$ *then go to Step* 5. *Else, set* $N = N_0$, $N_0 = N_p$ *and go to Step* 3.
Step 5: **Optimal makespan** *Set* $C^*_{\max} = UB$. *Stop*.

6. Computational experiments

In this section we present an empirical analysis of the performance of the proposed branch-and-bound algorithm. The algorithm was coded in C and compiled with Visual C++ 5.0. The computational experiments were carried out on a Pentium IV 2.8 GHz Personal Computer with 1 GB RAM.

6.1. Test generation

We carried out a series of experiments on test problems that were randomly generated in the following way. The number of jobs n is taken equal to 50, 100, 150, 200, 300, 500, and 700. The number of machines m is taken equal to 2, 3, 5, 7, 10, and 20. The processing times are drawn from the discrete uniform distribution on [1,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, K\frac{n}{m}]$, where K is taken equal to 1, 3, 5 and 7. The availability times are drawn from the discrete uniform distribution on $[r_{\min}, r_{\max}]$, where $r_{\min}$ and $r_{\max}$ are the smallest and largest release dates, respectively. We combined these problem characteristics to obtain 168 problem classes. For each class, 10 instances were generated. A CPU time limit of 300 s was set for each run.

6.2. Performance of the algorithm

We found that our algorithm solved 1674 out of 1680 instances within the CPU time limit of 300 s. Moreover, it requires an average computation time of only 4.70 s. Tables 3 a–d provide the details of the performance of our algorithm according to the variation of K, n and m. In these tables, we provide:

- *Time*: mean CPU time (in s).
- *NN*: mean number of nodes.
- *US*: number of instances for which optimality was not proved after reaching the time limit. The values between parentheses denote the provided lower and upper bounds of unsolved instances.

We observe that all of the instances (except one) with $m \leqslant 7$ have been solved within the time limit of 300 s. For larger values of m, at least 9 out of 10 instances have been solved for each problem class. In particular, for $K \leqslant 3$, 18 out of 20 of the largest instances (700 jobs and 20 machines) have been solved, on average, in less than 1 min. Note that, for all of the unsolved instances, the absolute gap ($UB - LB$) is always equal to 1. The CPU time seems to be more sensitive to the variation of n than m. Also, we observe that several instances

Table 3

m	*n*	*Time*	*NN*	*US*	*m*	*n*	*Time*	*NN*	*US*
(a) Sensitivity to the variation of n and m for $K = 1$									
2	50	0.03	1.00	0	7	50	0.04	1.00	0
	100	0.11	1.00	0		100	0.09	1.00	0
	150	0.10	1.00	0		150	0.34	1.00	0
	200	0.45	1.00	0		200	0.65	1.00	0
	300	1.70	1.00	0		300	3.42	90.90	0
	500	7.93	1.00	0		500	9.68	1.00	0
	700	28.95	1.00	0		700	24.59	73.90	0
3	50	0.03	1.00	0	10	50	0.03	1.00	0
	100	0.12	10.90	0		100	0.16	22.40	0
	150	0.42	1.00	0		150	0.39	1.00	0
	200	0.98	23.40	0		200	0.63	1.00	0
	300	2.84	31.00	0		300	2.12	36.30	0
	500	8.49	54.70	0		500	14.99	116.00	0
	700	21.42	1.00	0		700	27.95	638.30	0
5	50	0.03	6.00	0	20	50	30.02	26761.20	1 (14–15)
	100	3.72	2648.70	0		100	0.13	1.00	0
	150	0.36	1.00	0		150	30.68	14847.00	1 (47–48)
	200	0.97	21.10	0		200	43.36	21658.10	0
	300	4.55	105.60	0		300	1.82	1.00	0
	500	10.91	54.40	0		500	8.26	53.30	0
	700	28.24	1.00	0		700	50.21	8757.10	1 (212–213)
(b) Sensitivity to the variation of n and m for $K = 3$									
2	50	0.03	1.00	0	7	50	0.05	5.90	0
	100	0.12	10.90	0		100	0.28	80.20	0
	150	0.34	1.00	0		150	0.77	126.40	0
	200	0.52	1.00	0		200	0.72	1.00	0
	300	2.33	1.00	0		300	2.97	80.70	0
	500	4.74	1.00	0		500	12.14	105.60	0
	700	17.47	1.00	0		700	20.37	1.00	0
3	50	0.05	11.10	0	10	50	0.07	12.50	0
	100	0.09	1.00	0		100	30.12	21038.70	1 (75–76)
	150	0.37	16.10	0		150	0.46	16.30	0
	200	0.49	1.00	0		200	0.72	1.00	0
	300	1.64	1.00	0		300	3.00	30.90	0
	500	7.67	1.00	0		500	9.54	1.00	0
	700	38.03	70.90	0		700	26.79	1.00	0
5	50	0.04	6.10	0	20	50	0.02	1.00	0
	100	0.12	1.00	0		100	0.11	1.00	0
	150	0.39	19.40	0		150	1.39	431.70	0
	200	0.64	1.00	0		200	0.73	1.00	0
	300	2.80	1.00	0		300	1.47	1.00	0
	500	7.97	1.00	0		500	8.77	1.00	0
	700	19.33	1.00	0		700	54.37	8512.50	1 (246–247)

Table 3 (*continued*)

m	*n*	*Time*	*NN*	*US*	*m*	*n*	*Time*	*NN*	*US*
(c) Sensitivity to the variation of n and m for $K = 5$									
2	50	0.01	1.00	0	7	50	0.01	1.00	0
	100	0.07	1.00	0		100	1.92	915.00	0
	150	0.26	1.00	0		150	0.07	1.00	0
	200	0.04	1.00	0		200	0.10	1.00	0
	300	1.21	1.00	0		300	0.60	1.00	0
	500	5.25	1.00	0		500	1.80	1.00	0
	700	13.47	1.00	0		700	4.73	1.00	0
3	50	0.01	1.00	0	10	50	0.01	1.00	0
	100	0.07	1.00	0		100	0.02	1.00	0
	150	30.08	18195.60	1 (533–534)		150	0.02	1.00	0
	200	0.30	1.00	0		200	0.10	1.00	0
	300	0.91	1.00	0		300	0.59	1.00	0
	500	2.50	1.00	0		500	0.59	1.00	0
	700	9.81	1.00	0		700	3.19	1.00	0
5	50	0.01	1.00	0	20	50	0.01	1.00	0
	100	0.04	1.00	0		100	0.01	1.00	0
	150	0.10	1.00	0		150	0.01	1.00	0
	200	0.24	1.00	0		200	0.01	1.00	0
	300	0.60	1.00	0		300	0.01	1.00	0
	500	1.20	1.00	0		500	0.01	1.00	0
	700	6.53	1.00	0		700	0.01	1.00	0
(d) Sensitivity to the variation of n and m for $K = 7$									
2	50	0.01	1.00	0	7	50	0.01	1.00	0
	100	0.06	1.00	0		100	0.01	1.00	0
	150	0.12	1.00	0		150	0.09	1.00	0
	200	0.22	1.00	0		200	0.24	1.00	0
	300	0.83	1.00	0		300	0.28	1.00	0
	500	5.78	1.00	0		500	3.57	1.00	0
	700	12.71	1.00	0		700	3.22	1.00	0
3	50	0.01	1.00	0	10	50	0.01	1.00	0
	100	0.04	1.00	0		100	0.02	1.00	0
	150	0.08	1.00	0		150	0.01	1.00	0
	200	0.24	1.00	0		200	0.04	1.00	0
	300	0.57	1.00	0		300	0.14	1.00	0
	500	2.80	1.00	0		500	0.58	1.00	0
	700	9.82	1.00	0		700	0.01	1.00	0
5	50	0.01	1.00	0	20	50	0.01	1.00	0
	100	0.03	1.00	0		100	0.01	1.00	0
	150	0.11	1.00	0		150	0.05	1.00	0
	200	0.25	1.00	0		200	0.01	1.00	0
	300	0.44	1.00	0		300	0.01	1.00	0
	500	1.71	1.00	0		500	0.01	1.00	0
	700	7.72	1.00	0		700	0.01	1.00	0

have been solved at the root node, which suggests the effectiveness of the proposed bounds. In particular, all of the instances with $K = 7$ have been solved at the root node.

At this point, it is worth noting that we performed similar computational experiments with two additional variants of our branch-and-bound algorithm which are based on LB_1 and LB_2, respectively. We found that these two variants have a very similar behavior for all problem classes, and that they are largely outperformed by the branch-and-bound algorithm which is based on LB_3. Indeed, they require about 5 times more CPU time, explore about 41 times more nodes, and fail to solve 16 times more instances. The only significant exception being the set of instances with $K = 1$ where all of the three algorithms require comparable CPU time.

6.3. Performance on particular cases

In this section, we provide the analysis of additional experiments that have been carried out in order to assess the performance of our algorithm on the two special cases $P, NC_{inc}||C_{\max}$ and $P|r_j, q_j|C_{\max}$.

6.3.1. Performance on the $P, NC_{inc}||C_{\max}$

To the best of our knowledge, no exact algorithm has been so far proposed in the literature for $P, NC_{inc}||C_{\max}$. In order to assess the performance of our algorithm on this special case we generated a set of 1680 instances in the same way as described in Section 6.1, but with heads and tails equal to zero, and machine availabilities drawn from the discrete uniform distribution on $[1, K\frac{n}{m}]$ $(K = 1, 3, 5, 7)$.

Table 4 shows that our algorithm performs remarkably well. Indeed, only one instance out of 1680 has not been solved within the time limit of 300 s. Moreover, it requires, on average, less than 30 s to solve large-sized instances with 700 jobs and 20 machines.

6.3.2. Performance on the $P|r_j, q_j|C_{\max}$

First, we compared our algorithm with the two time-window-based B&B algorithms proposed in [7] for $P|r_j, q_j|C_{\max}$. In order to obtain meaningful results, we tested our algorithm on the same set of 720 problem tests generated by Gharbi and Haouari [7]. Moreover, the runs were carried out on the same Pentium III 733 MHz Personal Computer used in [7]. The results are depicted in Table 5 . In this table, TW_1 and TW_2 denote the two time-window-based algorithms described in [7], and A denotes our $B\&B$ algorithm.

Table 5 provides strong evidence that our algorithm consistently outperforms TW_1 and TW_2. Indeed, we observe that it solved all of the 720 instances within the time limit of 300 s. It is worth noting that the *maximal* computing time of our algorithm is only 30.54 s. The mean CPU time being 1.25 s, this algorithm is 14.84 times faster than TW_1 and 21.82 times faster than TW_2. Moreover, the number of nodes explored is, on average, 128.55 times less than the number of nodes explored by TW_1 and 57.35 times less than the number of nodes explored by TW_2. Table 6 depicts the sensitivity of our algorithm to the variation of K. As it can be seen from this table, the problems become easier as K increases. It is worth noting that the two variants of our algorithm which are based on LB_1 and LB_2, respectively, exhibit a very poor performance on these $P|r_j, q_j|C_{\max}$ instances. Indeed, both algorithms

Table 4
Performance on $P, NC_{inc}||C_{\max}$ instances

m	n	*Time*	*NN*	*US*	m	n	*Time*	*NN*	*US*
2	50	0.02	1.00	0	7	50	0.04	4.67	0
	100	0.08	3.47	0		100	0.15	10.90	0
	150	0.27	8.45	0		150	0.45	19.62	0
	200	0.63	1.00	0		200	1.06	30.85	0
	300	1.56	8.47	0		300	2.94	38.37	0
	500	7.01	13.47	0		500	12.40	50.90	0
	700	17.01	1.00	0		700	27.37	1.00	0
3	50	0.03	3.45	0	10	50	0.04	7.37	0
	100	0.12	5.95	0		100	0.13	5.95	0
	150	0.32	4.72	0		150	0.33	4.72	0
	200	0.74	10.95	0		200	0.85	5.97	0
	300	2.63	23.42	0		300	2.66	8.47	0
	500	9.22	13.47	0		500	9.26	13.47	0
	700	26.60	18.47	0		700	27.49	53.42	0
5	50	0.03	3.45	0	20	50	7.62	4884.15	1 (19–20)
	100	0.15	1.00	0		100	0.18	11.92	0
	150	0.46	19.62	0		150	0.39	15.90	0
	200	0.94	10.95	0		200	0.95	30.85	0
	300	2.27	8.47	0		300	2.83	45.85	0
	500	9.47	13.47	0		500	9.92	63.37	0
	700	35.36	35.95	0		700	25.81	53.42	0

Table 5
Performance on $P|r_j, q_j|C_{\max}$ instances

	Time	*NN*	*US*[a]
TW_1	18.56	750.78	5.83
TW_2	27.28	334.96	7.50
A	1.25	5.84	0.00

[a] Here US denotes the *percentage* of unsolved instances.

Table 6
Sensitivity of A to the variation of K for $P|r_j, q_j|C_{\max}$ instances

K	*Time*	*NN*
1	2.59	14.30
3	1.28	7.06
5	0.81	1.00
7	0.30	1.00

Table 7
Performance on large-sized $P|r_j, q_j|C_{\max}$ instances

m	*n*	*Time*	*NN*	*US*
6	400	6.69	28.50	0
	600	22.89	157.45	0
	800	50.15	181.80	0
	1000	79.84	51.15	0
7	400	13.37	69.25	0
	600	20.97	62.05	0
	800	59.48	122.25	1 (630–631)
	1000	91.58	245.70	0
8	400	5.97	140.25	0
	600	17.62	147.50	0
	800	44.92	139.00	0
	1000	98.21	108.60	1 (698–699)
9	400	5.73	23.45	0
	600	23.46	1832.75	0
	800	33.70	1.00	0
	1000	66.72	229.80	0
10	400	4.55	1.00	0
	600	16.49	122.45	0
	800	47.86	652.55	0
	1000	99.82	635.95	1 (545–546)

require about 15 times more CPU time and explore about 7000 times more nodes than the algorithm which is based on LB_3 does.

Moreover, we run our algorithm on larger test problems generated as follows. The number of jobs *n* is taken equal to 400, 600, 800, and 1000. The number of machines *m* is taken equal to 6, 7, 8, 9 and 10. The processing times are drawn from the discrete uniform distribution on [2,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, n/m]$. It is worth noting that, according to the analysis of Carlier [3] and Gharbi and Haouari [7], this set of instances belongs to the hardest class ($K = 1$ and large number of machines). For each combination of *n* and *m*, 20 instances are generated. The CPU time limit was kept equal to 300 s.

Table 7 shows that only three instances out of 400 have not been solved within the time limit. It is worth noting that while TW_1 and TW_2 experience difficulties in solving instances of 300 jobs and 4 machines of this hard class (40% of unsolved instances by TW_1 [7]), our algorithm makes it feasible to solve large-sized instances with up to 1000 jobs and 10 machines in about 100 seconds, on average.

References

[1] P. Baptiste, S. Demassey, Tight LP bounds for resource constrained project scheduling, OR Spektrum 26 (2004) 251–262.

[2] P. Baptiste, C. Le Pape, W. Nuijten, Satisfiability tests and time bound adjustments for cumulative scheduling problems, Ann. Oper. Res. 92 (1999) 305–333.
[3] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European J. Oper. Res. 29 (1987) 298–306.
[4] J. Carlier, B. Latapie, Une méthode arborescente pour résoudre les problèmes cumulatifs, RAIRO 25 (1991) 311–340.
[5] J. Carlier, E. Néron, An exact method for solving the multiprocessor flowshop, RAIRO-Oper. Res. 34 (2000) 1–25.
[6] J. Carlier, E. Pinson, Jackson's pseudo preemptive schedule for the $Pm/r_j, q_j/C_{\max}$ scheduling problem, Ann. Oper. Res. 83 (1998) 41–58.
[7] A. Gharbi, M. Haouari, Minimizing makespan on parallel machines subject to release dates and delivery times, J. Schedul. 5 (2002) 329–355.
[8] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[9] M. Haouari, A. Gharbi, An improved max-flow based lower bound for minimizing maximum lateness on identical parallel machines, Oper. Res. Lett. 31 (2003) 49–52.
[10] H. Hoogeveen, C. Hurkens, J.K. Lenstra, A. Vandevelde, Lower bounds for the multiprocessor flow shop, Second Workshop on Models and Algorithms for Planning and Scheduling, Wernigerode, 1995.
[11] W.A. Horn, Some simple scheduling algorithms, Naval Res. Logistics Quart. 21 (1974) 177–185.
[12] H. Kellerer, Algorithms for multiprocessor scheduling with machine release times, IIE Trans. 30 (1998) 991–999.
[13] H. Kellerer, R. Mansini, U. Pfersch, M.G. Speranza, An efficient fully polynomial approximation scheme for the subset-sum problem, J. Comput. System Sci. 66 (2003) 349–370.
[14] J. Labetoulle, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to release dates, Progress in Combinatorial Optimization, Academic Press, Florida, 1984, pp. 245–261.
[15] A. Lahrichi, Ordonnancements: La Notion de "Parties Obligatoires" et son Application aux Problèmes Cumulatifs, RAIRO-RO 16 (1982) 241–262.
[16] C.-Y. Lee, Parallel machine scheduling with non simultaneous machine available time, Discrete Appl. Math. 30 (1991) 53–61.
[17] C.-Y. Lee, Y. He, G. Tang, A note on parallel machine scheduling with non simultaneous machine available time, Discrete Appl. Math. 100 (2000) 133–135.
[18] P. Lopez, J. Erschler, P. Esquirol, Ordonnancement de Tâches sous Contraintes: une Approche Energétique, RAIRO-APII 26 (1992) 453–481.
[19] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.
[20] E. Néron, P. Baptiste, J.N.D. Gupta, Solving hybrid flow shop problem using energetic reasoning and global operations, Omega 29 (2001) 501–511.
[21] M. Perregaard, Branch-and-bound method for the multiprocessor jobshop and flowshop scheduling problem, Master Thesis, Datalogisk institut KØbenhavns Universitet, 1995.
[22] D. Pisinger, Dynamic programming on the word RAM, Algorithmica 35 (2003) 437–459.
[23] G. Schmidt, Scheduling with limited machine availability, European J. Oper. Res. 121 (2000) 1–15.
[24] N.Y. Soma, P. Toth, An exact algorithm for the subset sum problem, European J. Oper. Res. 136 (2002) 57–66.
[25] A. Vandevelde, Minimizing the makespan in a multiprocessor flow shop, Master's Thesis, Eindhoven University of Technology, The Netherlands, 1994.
[26] S.T. Webster, A general lower bound for the makespan problem, European J. Oper. Res. 89 (1996) 516–524.